March 1981

Report. No. STAN-CS-81-846

The Byzantine Generals Strike Again

by

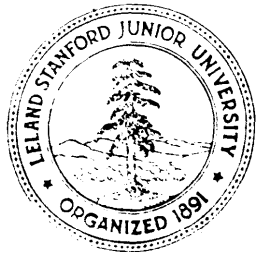
Danny Dolev

Research sponsored in part by

National Science Foundation

Department of Computer Science

Stanford University
Stanford, CA 94305



The Byzantine Generals Strike Again

Danny Dolev
Computer Science Department
Stan ford University
Stanford, CA 94305

Abstract

Can unanimity be achieved in an unreliable distributed system? This problem was named "The Byzantine Generals Problem," by Lamport, Pease and Shostak [LSP80]. The results obtained in the present paper prove that unanimity is achievable in any distributed system *if and* only *if* the number of faulty processors in the system is:
1) less than one third of the total number of processors; and
2) less than one half of the connectivity of the system's network.
In cases where unanimity is achievable, algorithms to obtain it are given. This result forms a complete characterization of networks in light of the Byzantine Problem.

This work was supported in part by Chaim Weizmann Postdoctoral Fellowship and in part by National Science Foundation grant MCS-80-12907.

1. Introduction

Unanimity in an unreliable distributed system is still far from being well understood. The major task is to circumvent errors without losing unanimity. This can be achieved if all the reliable members of the system agree upon the content of the messages in the system, especially those messages corresponding to the faulty parts of the system, even where the faulty parts cannot be uniquely identified.

The assumption is that a faulty processor can do whatever it likes. Thus, a faulty processor can behave very strangely: It can alter the information relayed through itself; it can block such information from being relayed; it can incorrectly reroute the information, and in the worst case, it can send conflicting information to different members of the system.

Some of the processors may consider a faulty processor to be a reliable one and a reliable processor to be faulty. Obviously, there is a limit to the number of faulty processors a system can tolerate. The problem of achieving unanimity is further complicated by the following compound question: Under what conditions does unanimity remain valid and what is the system's unanimity threshold'?

In the general case one does not know which processors are faulty. Moreover, in most cases one will never be able to know. To understand the reason for this, picture yourself as a processor in a distributed system that receives a message from a processor *z.* Assume that you want to make sure that *z* is reliable. So, you inquire what are the messages ths rest of the system received. You find out that the message you have received differs from all the rest. Is *z* a faulty processor? The answer is not necessarily positive. The possibility that the only reliable processors in the system are you and z always exists.

To overcome this logical pathology you should make your decision assuming an upper bound on the number of faulty processors in the system. That is, if the system contains more faulty processors than that upper bound, it will no longer matter what decision you are going to make. Throughout this paper *t* denotes the upper bound on the number of faulty processors in the system. Knowing *t* a receiver can sometime determine that the sending processor is unreliable. This can be achieved in cases of "too many" conflicting versions of the sender's message. In such cases the receiver is said to "explicitly know" that the sending processor is faulty. In the next section a precise definition of these notions will be given.

Unfortunately, in many cases the faulty processors can still remain anonymous and undistinguishable from the reliable ones. Consider first the problem of a processor, say z, that sends its message to every other processor in the system. If more than one message is received, the reliable processors have to determine and agree upon which message was actually sent by z. Two types of agreements which can be achieved in the presence of faulty processors will be studied in this paper.

Agreement A:

A1) All the reliable processors that do not explicitly know that z is faulty agree on the same message.

A2) If *z* is reliable, then all the reliable processors agree on its message.

Agreement B:

B1) All the reliable processors agree on the same message.

B2) If z is reliable, then all the reliable processors agree on its message.

Agreement B was named the Byzantine Generals problem by Lamport, Shostak and Pease [LSP80]. Here it will be refered to *as* the *Byzantine Agreement*. For consistency Agreement A will be called the *Crusader Agreemenf.*

A characterization of a network's tolerance to faults for a given type of agreement is t, the upper bound on the number of faults a network can sustain and still reach the agreement. The upper bound t should depend on the topology of the network. For example, if the network graph is a tree, it is clear that almost every faulty processor can prevent the system from achieving both the Crusader and the Byzantine Agreement.

The first work toward finding the threshold t for some networks was done by Pease, Shostak, and Lamport and is described in their paper "Reaching Agreement in the Presence of Faults" [PSL80]. In that paper and in an extended one, "The Byzantine Generals Problem*' [LSP80], the authors analyze the problem of reaching an agreement of type B, the Byzantine Agreement, in a complete network in which every pair of processors are directly connected. They present an algorithm to circumvent up to $(n\text{-}1)/3$ faulty processors, where n is the number of processors in the system. They proved that this threshold is tight, that is one cannot guarantee a unique agreement if the number of faulty processors is not less than a third. Lamport, Shostak, and Pease [LSP80] found another algorithm for a special type of networks that they call p-regular networks, but in this case the result is no longer tight. In a later paper Lamport [L80] studied the complete network with respect to another type of agreement called "The Weak Byzantine Generals Problem."

The results presented in the present paper provide a complete characterization of t for every network. The value of t is independent of the type of agreement and depends only on the number of processors and the connectivity of the network. The results can be stated as follows:

Both the Crusader and the Byzantine Agreements can be achieved in a network G if and only if:

1) t is less than one half of the connectivity of G; and

2) t is less than one third of the total number of processors in G.

The fact that both agreements are bounded by the same threshold t suggests the following (nontrivial) exercise:

> *Assuming you know how to reach the Crusader Agreement, can you reach the Byzantine Agreement?*

This exercise is left to the reader to develop a better understanding of the issue.

Note that the Crusader Agreement implies that if the sending processor z is faulty, then the network contains three sets of processors: first, the set of the faulty processors; second, the set of the reliable processors that explicitly know that z is a faulty processor; third, the set containing the rest of the reliable processors, those that agree on the same message. To obtain the Byzantine agreement the reliable processors may only exchange their knowledge. Beware, the exercise is not as easy as it looks, The algorithm for the Byzantine Agreement described later in this paper is independent of that of the Crusader Agreement.

2. Basic Notions and Assumptions

For clear representation of the relationships among the processors the following notions are used. A sender of messages is called the *transmitter* and the messages it sends carry its *value.* The transmitter sends its values to its *receivers* either directly or through other processors called *relays.* A processor can be a transmitter, a receiver or a relay according to its function in the network with respect to a given message. A processor is *reliable* if it transfers the messages it has received without altering or eavesdropping on them. A *reliable transmitter* is a reliable processor that sends the same value to all its receivers. A *faulty processor* is a relay or a transmitter that is not reliable. Furthermore, assume that a reliable processor does not read a message that it has to relay to someone else. This extra assumption does not change the conditions for achieving the agreements.

Let us concentrate on the case of a single processor that sends its value to the rest of the network. Solving this case successfully will enable us to handle the general case in which every processor sends its own value to every other processor.

For simplicity assume that every message contains the information about the route through which it is supposed to be delivered. Thus, before sending a message the transmitter chooses a route and sends the message containing the route. The receiver, however, does not know in advance the route through which it is going to receive the message. Notice that a faulty processor may also change the routing through which the message is supposed to be delivered. Moreover a faulty processor may also produce many false copies of the message it is supposed to relay, then send them through various routes of its own choice.

A faulty processor may change the record of the route to prevent the receiver from identify ing it as the source of faulty messages. To ensure the inclusion of faulty processors' names in the routes, assume that after a reliable processor receives a message to relay, it makes sure the pocessor from which the message has arrived is supposed to relay the message to itself. Only then does it relay the message to the next processor along the route to the receiver.

Throughout all the above discussion one issue was omitted: synchronization. Consider an asynchronous distributed system. In a reliable asynchrnous system there should be an upper bound on the amount of time between sending and receiving of a message. If the transmitter sends several messages to the same receiver, then, in a reliable system, the above upper bound determines the maximum length of the time-interval during which all the messages should arrive. In an unreliable system every message that has not arrived on time can be ignored, because it was relayed by a faulty processor.

Summary of assumptions:

1. The processors are arranged in a *k* connected bidirectional network.
2. Every processor knows the topology of the network.
3. Processors only communicate by sending messages along links.
4. Every processor can identify the neighbor from which it recieves each message.
5. Every message includes the route through which it is suppose to be delivered.
6. A reliable processor relays a message to its neibour only if the neigbour appears after itself in the message's route.
7. A reliable processor relays a message only if the processor from which it received the message appears immediatly before itself in the message's route.
8. A reliable processor relays messages without altering them and without eavesdropping on their values.
9. There exists an upper bound on the delay of relaying a message by a reliable processor.
10. There exists an upper bound, *t*, on the number of faulty processors in the system.

The ability of a faulty processor to disrupt may appear unlimited, but the situation is not so bad. Due to the Menger Theorem [H72], if the connectivity of the graph is k, then there exist at least *k* disjoint paths between every pair of processors. This shows that the transmitter has the ability to transmit *k* copies of its value through *k* disjoint routes to every receiver.

The following algorithm provides the receiver a method to purify the correct values out of all the copies it has received. Define "O" to be the zero value, which means that either the transmitter did not send any value or that there is no way to find a unique value out of the set of copies. This last case arises when the transmitter is a faulty processor that has sent too many conflicting values.

Definition: Let $\{a_1, \ldots, a_r\}$ be the set of copies of the transmitter's value received by the receiver x. Let U_x be a set of processors that does not contain the transmitter itself. A set U_x is called a set of *suspicious processors* determined by x if every message a_i that did not pass through processors in U_x carries the same value.

Algorithm *Purifying* $(t; a_1, \ldots, a_r; x)$

1. If a set U_x of up to t suspicious processors exists, then the purified value is the value of the messages that did not pass through U_x. If no message is left, the value is 0.
2. If there is no set U_x of cardinality up to t, then the purified value is 0.

Notice that if more than one set of suspicious processors exists, then there may be many purified values. But because of the way the algorithm will be used a plurality of possible values will not be a problem.

Before proving that the Purifying Algorithm actually does the right filtration, consider application of the Purifying Algorithm to the network shown in Figure 1.

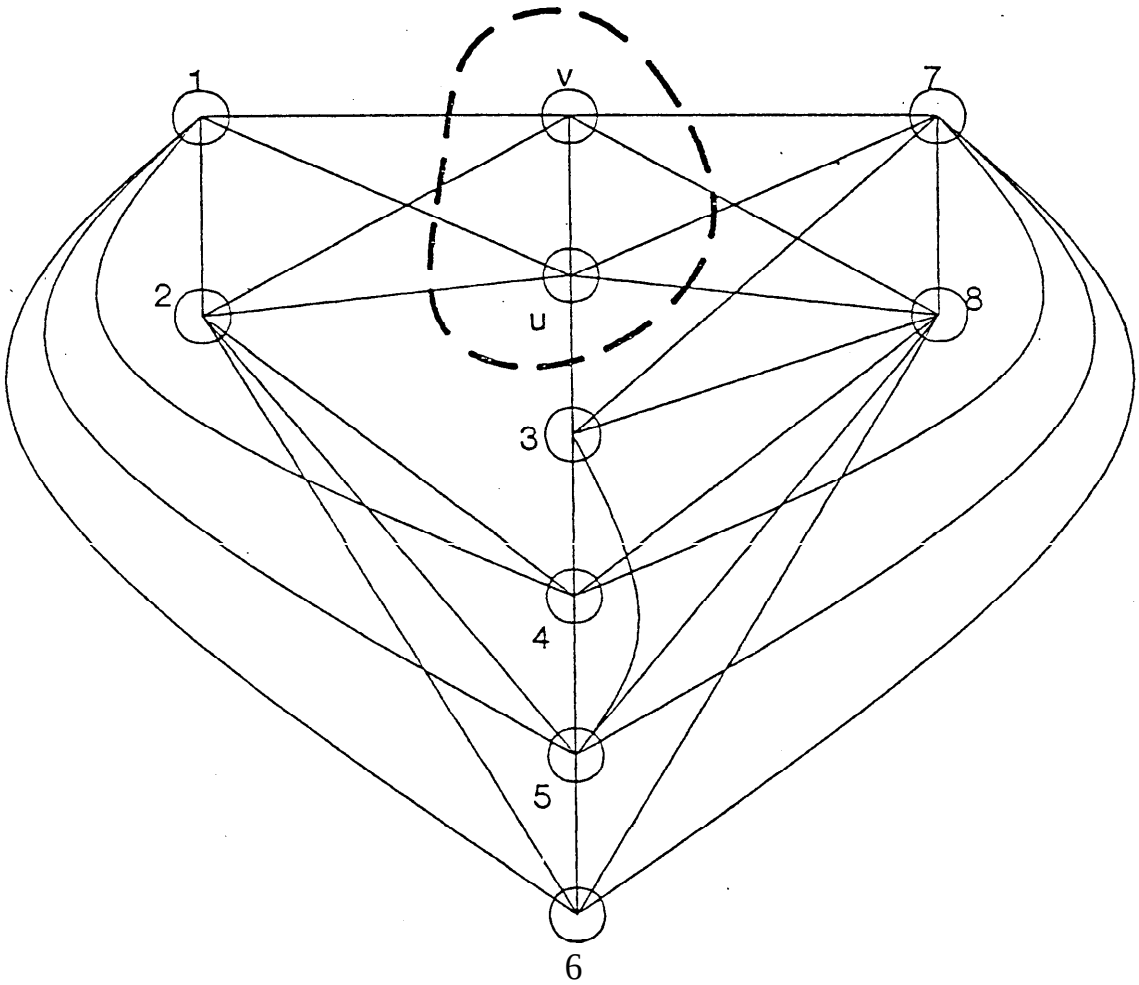


Figure 1: Ten processors with two faulty, v is the transmitter.

The network contains 10 processors and at most 2 faulty ones. Assume that v and *u* are the faulty processors. The transmitter v sends the value *a* to receivers 1 and 2 and the value *b* to the other receivers. Assume that Receiver 1 receives v 's value through the following paths:

1) a- *vl*

2) a: v21

3) a: vu1

4) b: v741

5) b: v851.

The Purifying Algorithm provides the purified value *a* to Receiver 1, by choosing {7,8} as the set of suspicious processors. Similarly, Receiver 2 obtains the value a. But the rest of the network obtain the value *b* by choosing {1,2} as the set of suspicious processors.

The following theorem proves that with sufficient connectivity all the reliable receivers obtain the same value if the transmitter is reliable.

Theorem 1:

Let G be a network of processors which contains at most t faulty processors, and the connectivity of which is at least $2t+1$. If a reliable transmitter transmits $2t+1$ copies of its value to every receiver, through disjoint paths, then, by use of the Purifying Algorithm, every reliable receiver can obtain the transmitter's value.

Proof: The reliable transmitter sends every receiver $2t+1$ copies of a value, through disjoint paths. It sends the same value to all recievers. Let $\{a_1, \ldots, a_r\}$ be the set of all the copies of the transmitter's value that receiver x receives. There are at most t faulty processors; therefore at most t values might be lost. This implies that the number of copies, r, is at least $t+1$. At least $t+1$ of the messages are relayed through routes which contain only reliable processors; each one of the reliable processors relays the message faithfully without changing it. This implies that at least $t+1$ of the received copies carry the original value. Note that if the transmitter were a faulty processor then the above reasoning would fail to hold.

It may be that the number of copies received is much more than $t+1$ and even that the majority of them carry a faulty value. The task of receiver x is to find the 'correct

value out of this mess. It does this by applying the Purifying Algorithm. Observe that the technique, described at the begining of the section, of adding the names of the processors along the route to the message; enables x to differentiate among the values. Every message which passed through faulty processors contains at least one name of a faulty processor; more precisely, every list of processors added to a message contains at least the name of the last faulty processor that relayed it.

Step 1 of the Purifying Algorithm requires one to look for a set Ux of up to t processors with the property that all the values which have not been relayed by processors from this set are the same. The network contains at most t faulty processors, and by definition, the transmitter is not one of them. Therefore receiver x should be able to find such a set U_x; it may be that the set it finds is not exactly the set of faulty processors, but U_x necessarily eliminates the wrong values. The set Ux cannot eliminate the correct values because there are at least $t+1$ independent copies of them and U_x can eliminate at most t independent copies. This completes the proof of the theorem. □

In the case where the transmitter is faulty, Theorem 1 does not ensure the ability to reach a unique agreement on a value. This case is discussed in the rest of the paper. Throughout the rest of the paper, whenever a transmitter sends its value it sends $2t+1$ copies through disjoint paths it chooses. Every receiver uses the Purifying Algorithm to find the value the transmitter wants it to receive.

The copies of values a receiver receives are evidence of the value sent by the transmitter. A reliable transmitter sends the same value to every processor. This implies that all further information about the transmitter's value that a receiver obtains from other receivers is also evidence of the transmitter's value. That is, if receiver y sends to receiver x a message saying "the transmitter has sent me the value a," then this message is also evidence to receiver x of the transmitter's value. Define the *set of evidence* to be the set of all the messages containing information about the

transmitter value, that a receiver receives. If the transmitter is reliable then similar reasoning to that of Theorem 1 implies that the purification of every set of evidence should uncover a set of suspicious processors.

Definition: Let $\{a_1, \ldots, a_r\}$ be a set of evidence of the transmitter's value received by receiver x. Receiver x *explicitly* knows that the transmitter is a faulty processor if it is unable to find a set of t suspicious processors.

Note that the definition implies that every set of t processors leave a set of copies which still contains conflicting values.

Lemma 2:
If the number of faulty processors is at most, t, then receiver x explicitly knows that the transmitter v is a faulty processor only if v is indeed a faulty processor.

Proof: Assume to the contrary that v is a reliable transmitter. Let T be the set of faulty processors, then its cardinality is at most t. In the subnetwork remaining after ignoring the set T there are no more faults. Therefore, every value x receives through this subnetwork, if it recieves anything, should be the transmitter's value.

Among all the candidate sets of suspicious processors that x checks it has to check T itself. But this leads to a contradiction because x cannot explicitly know that the transmitter is faulty. □

Lemma 2 implies that sometimes a faulty processor can be identified, because it has sent too many conflicting values. But even being known as a faulty processor by some of the processors does not prevent a faulty processor from being considered a reliable transmitter by others. Thus,' it can still prevent these processors from reaching an agreement on values.

3. The Necessary Conditions.

In term of the communication vocabulary defined in the previous section the *Crusader Agreement* becomes:

Crul) All reliable receivers that do not explicitly know that the transmitter is faulty agree on the same value.

Cru2) If the transmitter is reliable, then all reliable receivers agree on its value.

Similarly the *Byzantine Agreement* becomes:

Byzl) All reliable receivers agree on the same value.

Byz2) If the transmitter is reliable, then all reliable receivers agree on its value.

Thus, to prove the necessary condition what have to be shown is that if t is not less than half the connectivity or not less than third of the total number of processors then neither the Crusader Agreement nor the Byzantine Agreement can be achieved. It is enough to show that the Crusader Agreement cannot be achieved, because the Byzantine Agreement implies the Crusader Agreement.

Recall that n is the number of processors, t is the upper bound on the number of faulty processors in the system, and k denotes the connectivity of the network. Let the actual number of faulty processors is denoted by m. Thus $m \leq t$. If the number of processors is two or less then there is no problem reaching either agreement. Therefore assume that $n > 2$ and similarly that $m < n$.

The following example clarifies the difficulties of reaching an Agreement in the case $m \geq n/3$. Assume the network contains only three processors connected in a triangle and contains exactly one faulty processor. Let z be the transmitter and x be a processor which tries to follow the Crusader Agreement. Figure 2 desci-ibes the'

information x gets.

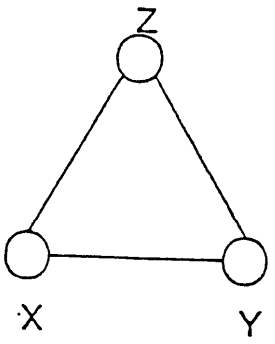


Figure 2: Three processors with one faulty processor.

Receiver x receives the value a directly from the transmitter and the different version of the transmitter's value, say b, from y. The receiver has to consider two possibilities:

P1) tie transmitter z is reliable and y is a faulty relay; and

P2) the transmitter z is faulty and y is a reliable relay.

Now, x can decide: "faulty transmitter" or it canchoose a value. It does not know if the situation is either P1 or P2. Therefore, if it decides "faulty transmitter" it might be that the situation is P1 which conflicts Cru2. This implies that it cannot decide "faulty transmitter" in this case. Processor x should choose the value a, otherwise it may fail to obey Cru2 in the case P1. Assume that the actual case is P2; therefore y as a reliable -processor faces the same problem: it receives b from z and a from x. The above argumentations imply that y will choose b to be the transmitter's value. These forced decisions contradict Crul. This proves that no matter what decision the processors make, the Crusader Agreement cannot be achieved.

Lemma 3:
No Crusader Agreement can be reached if there are only three processors which may contain faulty processors.

Proof: The nontrivial case is the example just studied; the rest of the cases are easy to prove. Note that the case m = 3 was excluded. □

The general case in which $n>3$ and $m>n/3$ is proved by reduction to the case of Lemma 3, using similar arguments to those appearing in [PSL80] and [LSP80]. A complete and formal proof-can be constructed similarly to those in [PSL80] and [L80].

Lemma 4:
No Crusader Agreement can be achieved in a network of *n* processors if the number of faulty processors is greater than or equal to a third of the total number of processor.

Proof: Assume to the contrary that there exists a network G with *n* processors and there exists an algorithm to achieve the Crusader Agreement in G for every distribution of up to t faulty processors in the network, where $t > n/3$. *The* impossibility of this situation will be proved by constructing the Crusader Agreement for a network of three processors, which contradicts Lemma 3.

The assumed algorithm should obtain the Crusader Agreement in the case where the actual number of faulty processors, say m, is not more than t . Divide the set of processors into three sets, X, Y and Z, such that each set contains at most t processors. Let Z be the set that includes the transmitter. Denote by H the reduced network obtained from G by identifying all the processors in each of the three sets. The new network H is a network of 3 processors, in which each processor represents up to t processors of the original network G. Reaching the Crusader Agreement in

the presence of one faulty processor in H can be simulated by reaching the Crusader Agreement in the network G, in the presence of up to t faulty processors. Similarly the assumed algorithm for reaching the Crusader Agreement in G can be simulated to reach the Crusader Agreement in H. But this contradicts Lemma 3. The constructed contradiction proves the lemma. □

The case in which the number of faulty processors is not less than half of the connectivity is easier to visualize, and is proved in Lemma 5. Figure 3 describes the case, schematically. The basic idea is that if the faulty processors are not less than half of the bottleneck they can prevent the reliable processors from reaching an agreement by behaving as a filter. Every message that passes from right to left would be changed to carry one value, and every message in the reverse direction would carry another value. This behavior can cause all the processors on the right side to agree on a different value from those on the left side.

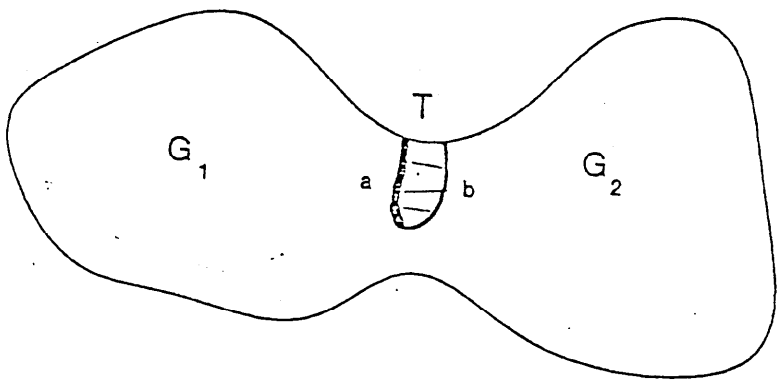


Figure 3.

Lemma 5:

. No Crusader Agrement can be achieved in a network of n processors if the number of faulty processors is not less than half of the connectivity of the network.

Proof: Let G be a network with connectivity k, and let $\{v_1, \ldots, v_k\}$ to be a set of

processors which disconnect the network to two nonempty parts, G_1 and G2. Assume that the subset $\{v_1, \ldots, v_t\}$ is the set of faulty processors, where $t \geq k/2$. Consider the following cases for the various places in which the transmitter can be.

Assume the transmitter v is in the subnetwork G-l, and that it sends the value a to all the receivers in the network. The faulty processors can follow the following doctrine: change every message which passes from G_1 to G2 to carry the value b; and leave every other value as a. Change the messages passing back from G_2 to G_1 to carry the value a. In this situation every receiver in G_1 can consider v to be a reliable processor and thus chooses a. Similarly the processors $\{v_{t+1}, \ldots, v_k\}$ choose a. But every receiver in G2 cannot consider v a faulty transmitter. They are able to ignore the conflicting values they have received by ignoring either the set $\{v_1, \ldots, v_t\}$ or $\{v_{t+1}, \ldots, v_k\}$. On the other hand, they cannot agree on a value because each of the values can be correct, depending upon what the transmitter has said and which processors are faulty. Since $t \geq k$ the receivers in G2 will choose b, in contradiction to Cru2. The case where the transmitter is in G_2 is identical by symetry.

Assume now that the transmitter is in the set $\{v_1, \ldots, v_k\}$. If the transmitter is reliable and sends the same value a to every processor, then by resonings, similar to the previous case, the faulty processors can prevent agreement. If the transmitter is faulty it can send the value a to G_1 and b to G2. The processors in G-l and G2 are not able to decide "faulty processor'* because in either case they are able to obtain a unique value. Thus, similarly to Lemma 3, in the case in which the transmitter is a faulty processor every decision implies violation of the Crusader Agreement. □

The necessary part of the main result is implied by the three lemmas above. The rest of the paper is the proof of the sufficiency for both agreements.

4. The Crusader Agreement

Although the Byzantine Agreement implies the Crusader Agreement it is better to study the Crusader Agreement the Byzantine agrement is achieved by a complex recursive algorithm. .

Algorithm CR(G, z, V, t) (the *Crusader Algorithm):*

1. The transmitter z sends its value to every receiver in V through $2t+1$ disjoint paths in G.
2. Each receiver applies the Purifying Algorithm on the values it receives from the transmitter to find the value the transmitter intended it to receive.
3. For every $u \in V$, let a_u be the value receiver u has obtained in step 2. Receiver u sends the value a_u to every other receiver in {V-u) through $2t+1$ disjoint paths.
4. Let M_u be the set containing the values receiver u has received in steps 1 and 3 and containing "O" for every value missing in step 3. Each receiver u finds a set U_u of t processors in {V-z} such that all the values in M_u which do not pass through processors in U_u are identical. If u is unable to find such a set, then it sets a_u to be *"faulty transmitter"*.

Before proving the validity of the above algorithm, study again the example described in Figure 1. The discussion in Section 2 implies that at step 2 of the Crusader Algorithm, Receiver 1 and Receiver 2 obtain the value a and all the rest of the Receivers obtain the value *b.* At step 3, every processor sends the value it has obtained to every other processor through 5 disjoint paths. At this step, processor 7 can learn that processors 1 and 2 claim to obtain the value *a,* and every other processor claims to obtain the value *b.* By choosing {1,2} to be the set of suspicious

processors it can obtain a unique value. Therefore, processor 7 decides that the value of the transmitter was *b*. By the same reasoning, processors 3 to 6 can also decide that *b* is the transmitter's value. But processors 1 and 2 are no longer able to find a set of up to two suspicious processors to achieve a unique value and therefore they have to decide "faulty transmitter." If the faulty processors v and *u* produced more faulty values or changed later values, then more processors would decide "faulty transmitter."

Lemma 6:
Let G be a network of at least $3t+1$ processors which contains at most t faulty processors, and the connectivity of which is at least $2t+1$. All receivers that do'not know explicitly that the transmitter is faulty agree upon the same value.

Proof If receiver x does succeed in finding a set U_x of up to t suspicious processors, then it does not know explicitly that the transmitter z is a faulty processor. Assume that processors x and y are such processors and therefore they choose ax and a_y, respectively, to be the values. Note that the case of receiving no value is also included.

Denote by T the set of faulty processors in the network G, thus $|T| \leq t$. Let U_x and U_y be the sets of suspicious processors choosen by x and y respectively. Each one of these two sets also contains at most t processors. The network contains at least $3t+1$ processors; therefore there exists at least one reliable processor w that is in neither set, that is, w is not in $T \cup U_x \cup U_y$. Denote by a_w the value w determines at step 2 of the algorithm; w is a reliable processor and therefore it sends the value a_w faithfully to all the other processors.

The network is at least $2t+1$ connected, therefore the subnetwork which does not contain T and U_x is at least one connected, which implies that there exists a path of reliable processors from w to x. Along such a path processor w sends its version of

the original value to processor x. This implies that x should get the correct value a,. Therefore, all the values it gets from the network which do not pass through U_x are equal to a,, which implies that $a_x = a_w$. Note that if w does not send anything, then every other processor chooses the value 0 as the value sent by w. Therefore a_x should also be 0.

The same argument implies that all the values that y gets are equal to a_w, which implies that $a_x = a_y$, which proves the lemma. []

Theorem 7: Let G be a network of at least $3t+1$ processors which contains at most t faulty processors, and the connectivity of which is at least $2t+1$. Algorithm CR(G,z,V,t) satisfies conditions Cru1 and Cru2.

Proof: Lemma 6 implies that Crul holds. The remaining case is the one where the transmitter is reliable. But here, by Theorem 1, every reliable processor gets the value sent by the reliable transmitter. Moreover, no processor can decide "faulty transmitter" because it could receive a wrong value from at most t processors. Lemma 6 implies that no value other than the transmitter's can be chosen. This completes the proof of the theorem. []

5. The Byzantine Algorithm.

To obtain the Byzantine Agreement the following Byzantine Algorithm is used. As a matter of fact, the actual result obtained is stronger: the algorithm enables us to achieve the Byzantine Agreement on any desired subnetwork of the underlying network. Throughout this section we use the following notation: Let G be a .

network of n processors with connectivity greater than $2t$; let $U \subset V$ be a subset of the set of processors of G; and let v be a processor of G not in U. The algorithm uses a function majority, which is a function every processor has for deciding what the value is for a given set of versions it has received. The function has to be such that it points out the majority value if there exists exactly one, otherwise it can be anything.

Algorithm BG(G, v, U, t, m) *(the Byzantine Algorithm)*:

1. The transmitter v sends its value to every receiver in U through $2t+1$ disjoint paths in G.
2. Each receiver applies the Purifying Algorithm on the values it receives from the transmitter to find the value the transmitter intended it to receive.
3. If $m>0$ then:

a. For every $u \in U$, let au be the value receiver u has obtained in step 2. Receiver u acts as the transmitter in the algorithm BG($G, u, U-u, t, m-1$) to send the value au to every other receiver in $U-u$.

b. For each $w \in U$ and each $u \neq w$ in U let $a_w(u)$ be the value receiver w receives from receiver u in step a. If no value is received, then set $a_w(u)=0$. Let $a_w(w)$ be the value receiver w has received from the transmitter v in step 2. Receiver w determines the value of v by $\underline{\text{majority}}\{a_w(x) | x \in U\}$.

To prove the validity of the algorithm BG observe that the same processor can be used again and again as a transmitter of disjoint paths between pairs of processors, even if it was a transmitter in previous recursions. Moreover, even being a faulty processor does not matter for the simple reason that the total number of paths that would be affected by faulty processors will never exceed t.

Before proving the validity of the algorithm, consider the example in Figure 4.

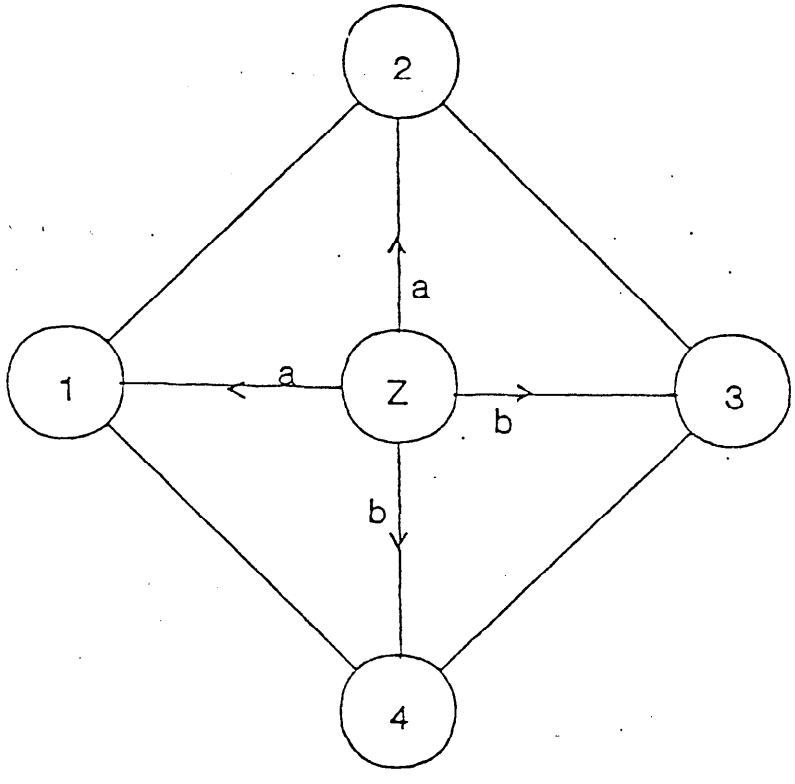


Figure 4: five processors and one faulty

The network contains 5 processorsand z is a faulty transmitter. At step 2 of the algorithm processors 1 and 2 obtain the value a while processors 3 and 4 obtain *b.* At step 3a every processor transfers its version to the rest. At step 3b every processor evaluates majority{*a,a, b, b),* which produces the same value.

Lemma 8:

Let G be a network of processors; U be a subset of processors from G with cardinality at least 2r-t m, and let v be a processor not in U. If the set of processors U contains at most r faulty processors, then Algorithm BG(G,v,U,t,m) satisfies condition Byz2 with respect to U. .

Proof: The condition Byz2 determines the behavior of the reliable processors in case the transmitter is reliable. The proof of the lemma is by induction on m. For $m=O$ Byz2 holds, since the transmitter is reliable, and by Theorem 1 every reliable receiver receives the same value.

Assume that the lemma holds for $m\text{-}1 \geq 0$. The reliable transmitter v sends its unique value a to every receiver in U by $2t+1$ disjoint paths. According to Theorem 1 every receiver w in U obtains in step 2 the value $a_w = a$. In step 3a, each processor u applies the algorithm BG($G,u,U\text{-}u,t,m\text{-}1$) to send the value it obtained to all the other processors in U-u. Since U contains at least $2r+m$ processors, then U-u contains at least $2r+m\text{-}1$ processors. This implies that the induction hypothesis can be applied . to reach the conclution that every other reliable processor in U receives from every other reliable processor the same value a. The set U contains at least $2r+m$ processors. Since $m>0$ and since there are at most r faulty processors in U the majority in step 3b, done by every individual receiver w, produces the same value a. This completes the proof of the lemma.0

The following theorem uses Lemma 8 to prove that algorithm BG(G,v,U,t,m) induces the Byzantine Agreement among the processors in U.

Theorem 9: Let G be a network of processors; U be a subset of processors from G with cardinality at least 3m, and let v be a processor not in U. If the set of processors $U \cup \{v\}$ contains at most m faulty processors, then Algorithm BG(G, v, U,t,m) satisfies conditions Byzl and Byz2 with respect to U.

Proof: The proof is by induction on m, the bound on the number of faulty processors in the set $U \cup \{v\}$. The case $m=O$ is the case where there are no faulty processors in U and v'is a reliable transmitter. In this case Theorem 1 implies that

the algorithm trivially satisfies Byzl and Byz2.

Assume that the theorem holds for *m-120.* Consider first the case in which the transmitter v is reliable. Lemma 8 holds for every distribution of faulty processors, especially for the case where $r=m$ faulty processors in U *U (v).* Therefore, algorithm BG satisfies Byz2 which implies Byzl for this case.

The only remaining case is that the transmitter is a faulty processor. There are at most *m* faulty processors in U U {v) and the transmitter is one of them, therefore U itself contains at most *m-l* faulty processors. Since U contains at least *3m* processors then *U-u* contains at least $3m-1 > 3(m-1)$ processors. This implies that the inductive assumption can be applied to step 3a. The validity of Byz2 and the inductive assumption on Byzl imply that in step 3b every two reliable processors get the same value for every processor in U. This leads to the fact that majority$\{a_w(x) | x \in U\}$ produces the same value for every processor w in U, which proves Byzl. □

Theorem 9 implies that the Byzantine Agreement, for the whole graph G, can be obtained by taking $V-\{v\}$ to be U and $t = m$.

Corollary 10:

The algorithm BG$(G, v, V-v, t, t)$ solves the Byzantine Generals problem for every network of more than $3t$ processors with connectivity greater than $2t$. □

6. Conclusion .

A characterization of distributed networks according to their ability to sustain arbitrary malfunctioning has been presented. The number of messages needed for achieving the Crusader Agreement is much less than that for the Byzantine Agreement. Lower bounds on the number of messages require further study.

The concepts and algorithms described can be employed in many aspects of distributed networks. Assumptions can be changed and agreements can be relaxed. For example, the assumptions that a network is synchronous, or that routes are predetermined can provide simpler algorithms. However, every relaxation ‘that leaves the faulty processors to do whatever they like requires the same bound on the number of faults.

Acknowledgments

Faulty algorithms to overcome faulty processors are very common. I wish to thank Michael Rodeh from IBM Israel for exploring the faults in my faulty algorithms, and helping me to gain the insight needed to solve the problem. I would like to express my gratitute to Uri Geva for helping me to present the nonfaulty algorithms in an understandable manner and to David Chaum, and Yonatan Malachi who have carefully read and commented on previous draft of this paper.

Refrences

[H72] F. Harary, *Graph Theory,* Addison-Wesley, 1972.

[L80] L. Lamport, "The Weak Byzantine Generals Problem," Technical Report 58, Computer Science Laboratory, SRI International, November 1980.

[LSP80] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem," Technical Report 54, Computer Science Laboratory, SRI International, March 1980.'

[PSL80] M. Pease, R. Shostak and L. Lamport, Reaching agreement in the presence of faults, *Journal **of the** ACM 27* (1980), 228-234.